3. Проектний розділ

В попередньому розділі було досліджено кілька шляхів побудови сховищ даних на основі платформи Hadoop. Результатом цього стало коротке порівняння двох наведених шляхів та структурні схеми обох наведених рішень.

В даному розділі буде спроектовано та детально описано список всіх компонентів системи та перелік технологій, обраних для реалізації. Потрібно брати до уваги факт, що технології в світі Hadoop розвиваються дуже стрімко і вже за короткий час весь перелік інструментів, обраних в даній роботі, може бути застарілим. Але навіть не зважаючи на це, таке спроектовано рішення дозволить підбирати та оновлювати всі частини системи шляхом заміни компонентів на більш нові. Платформа спроектована так, що інтеграція та оновлення компонентів системи є доволі легким і не протребує багато роботи розробника.

Далі структурна схема, розроблена в попередньому розділі, буде розширена деталями про логіку роботи та руху даних. Така схема зможе слугувати вихідною точкою для безпосередньої розробки програмного рішення і виконувати роль карти на шляху побудови системи.

1. Проектування сховища даних

В попередньому розділі було розроблено структурну схему сховища даних, призначеного для виконання аналітичних запитів на побудови аналітичних звітів. Було наведено призначення кожної структурної частини системи та способои інтеграції частин між собою. Тепер мною буде запропоновано низькорівневий опис роботи системи та перелік технологій для побудови такого сховища даних.

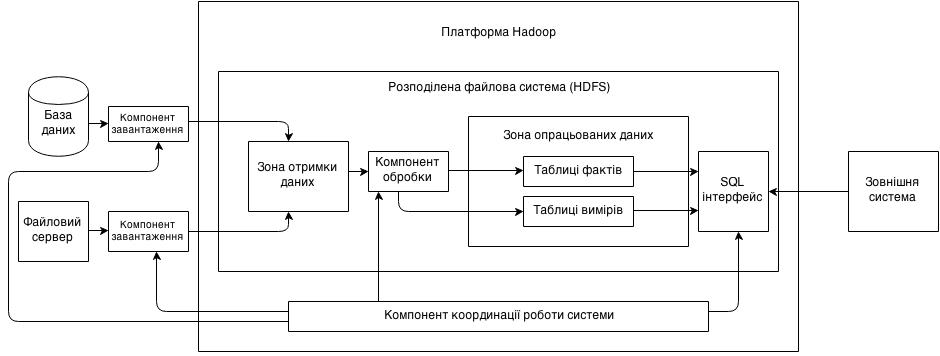


Рис. 3.1. Схема компонентів системи

В ході проектування мною буде обрано конкретні технології та інструменти для реалізації схеми наведеної на рис. 3.1. Як можна побачити, ця схема більш детально описує структуру системи, на відміну від схеми архітектури, наведеної в попередньому розділі. Основною різницею є наявність компонентів завантаження, обробки та координації роботи системи. Також в цій схемі детальніше наведено розмежування даних в розподіленій файловій системі. Таке розділення відображає логічні зони для даних, що існуватимуть в системі. В зоні отримання даних вся інформація зберігатиметься в такому вигляді, як вона надійшла з баз даних та файлових серверів. Згідно до структури побудови виміро-орієнтованих сховищ даних, всі дані будуть розбиті на таблиці фактів та таблиці вимірів, які і формуватимуть схему «зірка» або ж схему «сніжинка».

1. Вибір технологій для побудови системи

Оскільки система будуватиметься на основі платформи Hadoop, то одним з основих технологічних елементів, які потрібно обрати є дистрибутив Hadoop. Дистрибутивом платформи вважається пакет інтегрованих між собою та правильно налаштованих елементів екосистеми Hadoop. Сам по собі Hadoop це просто фреймворк для розподілених обчислень та розподілена файлова система. Всі елементи можна встановлювати самому, в ручному режимі, але це дуже ресурсоємкий процес, який потребує глибинних знань платформи. В даний момент є три найпоширеніших постачальника дистрибуцій: Hortonworks, Cloudera та MapR. З трьох описаних варіантів мною було вибрано постачальника Hortonworks. Ця компанія пропонує зовсім безплатно дистрибуцію HDP(Hortonworks Data Platform) 2.2. Цей набір включає в себе найновіші версії всіх основних компонентів екосистеми. Основними елементами, що входять до складу HDP 2.2 є:

* Hadoop & YARN(Yet another resource negotiator) 2.6.0
* Pig 0.14.0
* Hive & HCatalog 0.14.0
* HBase 0.98.4
* Spark 1.2.0
* Kafka 0.8.1
* Sqoop 1.4.5
* Ambari 1.7.0
* Oozie 4.1.0

Як видно зі списку – перелік технлогій доволі великий (а це тільки невелика частина), тому налаштувати їх вручну було б складно. Більшість наведених інстументів буде використовуватися при побудові системи. Далі буде наведено короткий опис призначення цих інструментів та спосіб, в який вони будуть використані при побудові сховища даних.

Основними компонентами, на яких власне й базуються всі обчислення в платформі є фреймворк Hadoop там менеджер ресурсів YARN. Це в першу чергу розподілена файлова система HDFS(Hadoop Distributed File System). Вона забезпечує надіє на ефективне зберігання великої кількості файлів на кластерах з великої кількості машин. Алгоритми розподілу даних в такій системі забезпечують велику швидкість доступу, читання та запису файлів. Файлова система також забезпечує високу найдійність за рахунок реплікакації даних. Менеджером ресурсів в платформі є компонент YARN. Він відповідає за розподіл виконання завдань по кластері з машин, та координує процес запуску, роботи, та збору результатів. Як HDFS так і YARN це набори процесів master та slave, які працюють на кожній з машин кластеру та спілкують з використанням мережних протоколів. Master-процеси виконують надсилання завданнь slave-процесам а також слідкують за їх виконанням. Всі обчислення, що відбувають в платформі Hadoop виконуються за посередництвом цих двох складових. Вони, на ряду з Map та Reduce завданнями є найбільш низькорівневими та фундаментальними блоками всіх Hadoop-систем.

Платформа Hadoop написана на мові Java та виконується як набір процесів у віртуальній машині Java. Це означає, що на цій мові можна написати будь яку програму, якщо в ній дотримана парадигма MapReduce, яка буде запускатися на кластері. Недоліком такого підходу є те, що як тільки логіка програми ставатиме порівняно тяжкою, буде надзвичайно важко відобразити її в термінах конструкцій мови Java. Для уникнення цих труднощів було розроблено кілька скриптових мов, які дозволяють набагато легше маніпулювати даними та виконувати їх перетворення. Однією з таких мов є скриптова мова Pig. В ній основними термінами є табличні структури даних та операції над ними. З допомогою цеї мови можна дуже легко будувати конвеєри обробки даних. Так з допомогої її конструкцій можна описати перелік трансформацій, які потрібно виконати над задекларованим набором даних, вказати джерело даних, місце та спосіб збереження та запустити на виконання. Транслятор цієї мови перетворить весь написаний код у набір Java MapReduce завдань, які будуть запущені на виконання в системі Hadoop. Використання такої мови значно пришвидшує і полегшує процес розробки. Особливо зручно використовувати такий інструмент, коли дані потребують достатньо складних перетворень, як наприклад зєднання по певному полю, перетин або фільтрація за певним критерієм. Процес розробки з використанням мови Pig є в кілька разів швидшим та потребує менше часу та зусиль на відлагодження.

Іншим схожим інструментом є Hive та HCatalog. Це дві взаємоповязані технології, що забезпечують зручний доступ до даних, що збережені в HDFS. HCatalog містить внутрішню базу даних з мета-інформацією про все, що знаходяться на HDFS. Ця інормація використовується при звертанні до даних та їх пошуку. Hive в свою чергу надає SQL-інтерфейс для структурованих даних в системі. Для забезпечення такого доступу, достатньо створити табицю в Hive, в які описати структуру даних та вказати місце розташування. Після цього ми зможемо легко звертатися до цих даних через SQL-інтерфейс. Як і у випадку Pig кожне звертання транслюється в набір Java MapReduce завдань та виконується на кластері. Виконання таких запитів значно поступається традиційним БД в швидкості, але це компенсовується обємами даних, на яких такі запити виконуються. На відміну від звичайних БД така система дозволяє виконувати складні SQL-запити на наборах даних, обємом в кілька десятків терабайт. Для оптимізації зберігання використовуються спеціальні формати з внутрішнім індексуванням та стисненням. На відміну від зберігання в звичайному текстовому форматі, таке зберігання дозволяє зменшити час доступу та обсяг необхідної для зберігання памяті в кілька десятків разів.

Іншим важливим компонентом, який, щоправда не буде використовуватися в цій реалізації системи є NOSQL база даних HBase. Вона забезпечує збереження та доступ майже в реальному часі до обємів даних в кілька десятків терабайт. Це база даних, в яій збереження даних організовано з групуванням по колонках. Характерною рисою її є те, що вона не потребує наперед заданої структури даних, що в ній зберігаються і підтримує її динамічну зміну. Через особливості внутрішньої організації, агрегуючі запити з використанням одної або кількох колонок в такій базі виконуються майже миттєво, незважаючи на обсяги інформації, що в ній збережена.

Компонентом, що все частіше використовується при побудові розподілених систем обробки інформації є фреймворк Spark. Це бібліотека функцій, які дозволяють виконувати обробку даних всіх даних безпосередньо в основній памяті. Як уже було описано в попередньому розділі, основною відмінністю цього інструменту від звичайних MapReduce завдань є широке використання основної памяті як для проміжних так і для основних результатів. Цей фреймворк володіє переліком АРІ для таких мов програмування як Java, Scala та Python, а це надає велику гнучкість в процесі розробки. Моніторинг та координація виконання Spark програм у Hadoop кластері виконується за допомогою менеджера ресурсів YARN. В даній реалізації ця технологія використовуватись не буде. Її використання буде доцільним при побудові системи, що дозволяє обробляти інформацію в реальному часі.

Для взаємодії з СКБД в платформі Hadoop існує інструмент Sqoop. Цей компонент екосистеми надає можливість проводити завантаження інформації з баз даних у розподілену файлову систему. Він володіє командною оболонкою, звертаннями до якої можна специфікувати які колонки з якої таблиці завантажувати в Hadoop. Одновним з основних завдань інструменту Sqoop є скоординувати кількість і частоту запиитів від машин кластера до сервера бази даних. Беручи до уваги, що кластер може складатися з десятків а то й сотень машин, то в ситуації коли всі ці машини почнуть виконувати запити до БД, вона може бути перевантажена і припинити роботу. Для уникнення цього кількість одночасних запитів контролююється даною технологією. Варто сказати, що Sqoop надає достатньо великий перелік можливих операцій для виконання. Так з його допомогою достатньо легко організувати інкременетальне завантаження або вивантаження певних таблиць з БД. В нашій системі цей інструмент буде використовуватися у ролі інтерфейсу для структурованих даних з СКБД. Його виклики відбуватимуться з shell-скриптів, або ж з допомогою функціоналу, який надає програма для координації виконання Oozie.

Для завантаження даних з файлових серверів в нашій системі буде використовуватися стандартний засіб linux для копіювання з віддалених хостів. Це утиліта scp. З її допомогою всі необхідні файли будуть скопійовані на один з компютерів розподіленої системи, а потім засобами файлових команд системи HDFS завантажуватимуться в неї. Такий підхід є простим і надійним. Існують певні програми та утиліти, що надають можливість більш детально конфігурувати цей процес, але для цілей нашої системи такого функціоналу цілком достатньо.

Платформа Hadoop є цілою екосистемою і складається з великої кількості допоміжних частин. Розгортання Hadoop кластера, на наборі машин є зовсім непростим завданням. Потрібно правильно налаштувати всі конфігурації кожної з підсистем. Також часто виникає проблема несумсності деяких версій компонентів між собою. Для спрощення цієї задачі існує інструмент, який дозволяє значно полегшити процес розгортання кластера. Це програма Ambari. Основним її призначенням є керування всіма сервісами, що відносяться до екосистеми Hadoop, та спрощення процесу розгортання кластеру. При її використанні достатньо вказати dns-імена всіх машин кластеру, задати топологію системи, вибрати які сервіси на які машини ми бажаємо встановити, а програма вже сама виконає установку на налаштування цих сервісів. Ця програма володіє зручним веб-інтерфейсом, що надає можливість в інтерактивному режимі керувати кожним сервісом та кожною машиною в межах кластера. Також її зручно використовувати для моніторингу стану системи. В користувацькому інтерфейсі відображається які сервіси запущені на кожній з машин і подається сигнал у випадку зупинки якогось з них. Цей інструмент буде використаний нами на самому початку побудови системи для розгортання та конфігурування кластера.

Одним з найважливіших пунктів при побудові складних систем є організації процесу запуску складних програм та конвеєрів даних. Рідко коли в складних системах обробка інформації обмежується запуском тільки одної підпрограми. Зазвичай таких підпрограм є дуже багато і послідовністю їх запуску відбувається за досить складною логікою. Так, деякі підпрограми повинні виконуватися тільки після завершення попередніх, а інші – з певними часовими інтервалами. Для координації таких послідовностей в екосистемі Hadoop існує інструмент Oozie. Він надає можливість будувати складні послідовності з виконання підпрограм, як наприклад Pig або Hive скриптів, Java програм для виконання MapReduce завдань, Sqoop команд і т.д. Характерною особливістю цього компонента є можливість конфігурування виконання з певними інтервалами, або ж при наявності специфікованих файлів. Це є дуже зручним, коли виникає необхідність запускати послідовності підпрограм тільки у випадку наявності тих чи інших даних. Oozie побудовано у вигляд клієнт-сервер програми. На деяких, або й на всіх машинах кластеру встановлюється Oozie-клієнт, а на одній – сервер. З клієнта на сервер відправляється опис послідовності виконання(у вигляді xml файлу) та параметри для цієї послідовності. Самі програми, які мають виконуватися в межах цієї послідовності повинні розміщуватися на HDFS а шлях до них має бути специфікований у конфігураціному файлі до послідовності виконання. Після отримання завдання, сервер вичитує необхідні параметри, звертається до програм та виконує їх запуск та координацію. Oozie володіє зручним веб-інтерфейсом. Ми можемо слідкувати за станом виконання програм та навіть керувати їх роботою (як наприклад зупиняти, або ж перезапускати певні її елементи). В нашій системі цей компонент використовуватиметься для побудови конвеєра даних. Він координуватиме послідовність та частоту запуску підпрограм в межах нашої програми.

Вся Hadoop платформа була спроектована для кластерів з установленими операційними системами сімейства Linux. Для нашої системи буде використано один з дистрибутивів сімейства Fedora – CentOS. Це безплатний та багатофункціональний дистрибутив. Він поширений в серверних системах та часто використовується в великих організаціях та промисловості. Ця операційна система є одною з найбільш поширених при розгортанні Hadoop кластерів. Для зручності її установки на великій кількості машин та щоб уникнути процесу ручної установку велику кількість разів буде використано інтсрумент автотиматизації та керування кластерами Vagrant. Це система яка значно спрощує процес керування та взаємодії з кластером. Вона дозволяє виконувати розгортання, запуск та зупинку великих кластерів майже миттєво, після їх налаштування. Це великою мірою зменшує час, що необіхний службовому персоналу для керування інфраструктурою системи. Як і всі вище згадані інструменти, цей програмний продукт є абсолютно безплатним, тому може бути використаним для побудови системи сховища даних.

1. Розробка алгоритму формування таблиць вимірів

Як вже було розглянуто в розділі 1 цієї МКР, в виміро-орієнтованих сховищах даних існує декілька видів таблиць вимірів. Вони відрізняються своєю структурою та порядком формування. В даній системі передбачається наявність компонентів, що виконують побудову шаблонів таблиць вимірів всіх трьох типів. Далі буде наведено алгоритми формування таблиць кожного з видів. Враховуючи що шляхи формування таблиць одного виду є схожі, не зважаючи від предметної галузі, то компонент, що виконує побудову таких таблиць може бути легко конфігурованим для подальшого використання при побудові незалежних сховищ даних.

В розділі 1 було наведено опис трьох типів повільно-змінюваних вимірів. Основна відмінність серед них полягає в переліку даних, що вони містять та правил запису в таку таблицю. Оскільки тип 1 не потребує ніякої алгоритмічної логіки при побудові таблиці, він не буде розглядатися в цій роботі.

Спершу розглянемо алгоритм побудови повільно-змінюваних вимірів типу 2. Нагадаю що в таблицях такого типу, при надходженні нового запису з унікальним ідентифікатором, що вже існує в таблиці, попередній запис має бути заміненим. Також має виконуватися дедублікація записів, тобто в таблиці немає бути записів з однаковими унікальними ідентифікаторами.

Дані для побудови повільно-змінюваних вимірів можуть надходити двома шляхами:

* Як підмножина полів в записі таблиці фактів;
* Як окремий набір даних, що прямо не звязаний з таблицею фактів.

Незалежно від того яким шляхом прийшли дані для побудови таблиці вимірів, вони володітимуть приблизно ідентичною структурою і саме таку структуру ми візьмемо як початкову точку для побудови таблиці вимірів. Структура інформації, що надійшла, обовязково має містити:

* Поле або набір полів, що унікально ідентифікують запис
* Поле або набір полів, які зберігають корисну інформацію (для якої власне і формується таблиця вимірів)

Можливим але не обовязковим є атрибут дати створення запису. Цей атрибут є необхідним для побудови таблиць вимірів типу 3 і також певною мірою спрощує побудову таблиць вимірів типу 2.

Алгоритм побудови таблиці вимірів другого типу з вхідних даних наведений в блок-схемі на рис. 3.2.



Рис. 3.2. Блок-схема алгоритму побудови таблиць вимірів другого типу

Як видно з блок-схеми на рис. 3.2. після завантаження всього набору даних першим кроком є групування записів по унікальному ідентифікатору. Результатом виконання цього кроку є набір груп записів з ідентичними ідентифікаторами. В межах однієї групи записи можуть як відрізнятися значеннями інших полів, так і бути ідентичними за цим критерієм. Кожна сформована група записів опрацьовується окремо. Якщо в такій групі присутній тільки один запис, то він одразу зберігається, без додаткових опрацювань. У випадку коли в групі є кілька записів, то спершу виконується порівняння по даті (у випадку якщо неможливо ідентифікувати з якого набору даних є конкретний запис – з тих що уже були збережені раніше, або ж з тих що надійшли). Якщо дати в кількох записах співпадають, то вибирається довільний з записів. У випадку, коли записи в групі відрізняються своїми датами створення, то вибирається той, в кого значення дати є найбільшим, тобто запис, що є новішим за інші. Після такої перевірки вибрані записи зберігаються в таблицю вимірів.

Процес формування повільно-змінюваних вимірів типу 3 є складнішим. Він потребує деяких маніпуляцій з полем часу створення запису, а також введення деякої службової інформації. Як було описано в розділі 1, виміри типу 3 зберігають всі значення кожного з записів, якщо вони змінюються на протязі певного часу. Тобто якщо в записі з унікальним ідентифікатором id1 спершу було значення value1, а через деякий час воно змінилось на value2 то ця зміна має бути збережена в таблиці. Також має бути можливість дізнатись яке значення було розміщено з ідентифікатором id1 в кожний конкретний момент після його створення. Ця інформація може бути корисна при побудові звітів за тривалі періоди часу, а також для збору статистики і відслідковування змін в певній частині предметної області.

Побудова такого виміру потребує одночасного опрацювання як даних що тільки надійшли, так і вже наявних в системі записів з таблиці даного виміру. Це необхідно для коректного встановлення часу використання значення кожного з записів.

Для того щоб отримати можливість звертатися до всіх попередніх значень полів з однаковим ідентифікатором в таблиці вводиться два службових поля – дата створення і дата завершення актуальності. Дата створення відображає час, коли дане значення запису було встановлене. Дата завершення актуальності показує час, коли це значення було змінене на нове. Таким чином, маючи такі додаткові дані ми можемо прослідковувати всю історію змін кожного з записів.

Алгоритм побудови таблиць вимірів типу 3 зображено на блок схемі на рис. 3.3.



Рис. 3.3. Блок-схема алгоритму побудови вимірів типу 3

Як видно з рис. 3.3. при побудові вимірів типу 3, над даними виконується ряд перетворень для того, щоб представити їх в необхідному вигляді. Після формування такого виміру, та таблиці фактів ми можемо тримати необхідний зріз збереженої інформації шляхом зєднання цих двох таблиць. Вибираючи дані з певними конкретними значеннями дат актуальності ми можемо отримати представлення збереженої інформації не тільки в розрізі конкретно обраного виміру, але й з привязкою до часу. Для прикладу, система може формувати аналітичний звіт з врахуванням значень, які були актуальні в певний період в минулому. Це дуже зручна і корисна властивість сховища даних.

В даному розділі не розглядається алгоритм перетворення даних, які належать таблицям фактів. Беручи до уваги те, що сховища даних є предметно орієнтованими, то й трансформації над кожними конкретними екземплярами інформації будуть значною мірою відрізнятися. Саме ця частина бізнес-логіка має бути винесена в окремий модуль, який для кожного окремого сховища даних буде створюватися окрема. Змінюючи та налаштовуючи цей компонент ми виконуватимемо конфігурацію всього сховища даних.

Подібна ситуація є також з частиною системи, яка забезпечує внутрішній та зовнішній інтерфейс системи. Ці інтерфейси повинні формуватися з врахуваннями специфіки та природи даних, назв та кількості полів записів. Як і компонент перетворення таблиць фактів, ці компоненти будуть відрізнятися для кожного конкретного екземпляра сховища даних і тому мають бути сконфігуровані вже безпосередньо в процесі розробки.

Послідовність формування таблиць фактів і таблиць вимірів досить рідко відрізняється в різних предметних областях, тому координатор виконання завдань в системі може повторно використовуватися. Також не залежить від предметної області й процес розгортання Hadoop кластера. Єдина різниця може в кількості машин, з яких складається кластер. Такі системи як Vagrant та Ambari дозволяють абстрагуватися від цієї проблеми та виконувати налаштування та запуск великих кластерів легко і швидко. Все що необхідно для цього – вказати dns-імена всіх машин та вибрати топологію кластера(кількість master- та slave-вузлів).

Як вже було сказано на початку розділу – вся основна програмна логіка буде написана з використанням скриптової мови Pig, яка в свою чергу транслюватиметься в Hadoop-завдання на основі Java-коду.

Система, що була спроектована в даному розіділі може бути розроблена та відлагоджена досить швидко з використанням обмежених розробницьких ресурсів. Проте вона потребує детального розуміння принципу роботи парадигми MapReduce для забезпечення оптимальності та достатньої продуктивності програмного рішення.